

# Tobii Pro Glasses 3

## Developer Guide



## Tobii Pro Glasses 3 Developer Guide

v1.7

April 2023

All rights reserved.

Copyright © Tobii AB (publ)

The information contained in this document is proprietary to Tobii AB. Any reproduction in part or whole without prior written authorization by Tobii AB is prohibited.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Content subject to change without notice.

Please visit [Tobii Connect](#) for updated versions of this document.

# Table of Contents

1 Introduction .....	5
1.1 Firmware version .....	5
1.2 Units and definitions .....	5
1.3 Document conventions .....	5
1.4 Tobii Pro Glasses 3 Example web client .....	5
1.5 Tobii Pro Glasses 3 .net SDK .....	6
1.6 Tobii Pro Glasses 3 python SDK .....	6
2 License agreement .....	7
3 API Concepts .....	8
3.1 WebSocket .....	8
3.2 HTTP .....	8
3.3 Objects .....	8
3.4 Properties .....	9
3.4.1 Read a property value .....	9
3.4.2 Write a property value .....	9
3.5 Actions .....	10
3.6 Signals .....	12
3.6.1 HTTP vs. WebSocket .....	12
4 Workflows .....	13
4.1 Discovery .....	13
4.1.1 Zeroconf .....	13
4.1.2 Hostname .....	13
4.1.3 mDNS .....	13
4.1.4 Wireless access point .....	13
4.2 Live view .....	14
4.2.1 WebRTC .....	14
4.2.2 RTSP .....	16
4.2.3 Render gaze in live view .....	16
4.3 Calibration .....	17
4.3.1 Calibration with calibration target .....	17
4.3.2 Calibration without calibration target .....	19
4.4 Make a recording .....	19
4.4.1 Start, stop, cancel .....	20
4.4.2 Visible-name vs. folder .....	21
4.4.3 Metadata in recordings .....	22
4.4.4 Recording events during a recording (FW 1.10+) .....	22
4.4.5 Take snapshots during the recording .....	24
4.4.6 Duration of the recording .....	24
4.5 Replay .....	24
4.5.1 Access recording data over HTTP .....	24
4.5.2 Replay a recording with RTSP .....	25
4.5.3 Replay a recording with WebRTC .....	25
4.6 Time synchronization and NTP .....	26
4.7 Time zone .....	27
4.8 Check status of the glasses .....	27

4.8.1 SD card status .....	27
4.8.2 Battery status .....	28
4.8.3 Firmware update status .....	28
4.9 Settings .....	28
4.9.1 Gaze frequency .....	28
4.9.2 Gaze overlay .....	29
4.10 Scene camera exposure (added in FW 1.29) .....	30
4.11 Zoom .....	31
5 SD card file system .....	34
6 Streams and timestamps .....	35
7 Network configuration .....	36
7.1 Ethernet .....	36
7.2 Wireless .....	36
8 Migrate from Pro Glasses 2 .....	39
8.1 Projects/participants/calibrations .....	39
8.2 Live view .....	39
8.3 Replay .....	39
8.4 Discovery .....	39
8.5 REST API .....	39
9 Firmware upgrade .....	40
10 TTL and sync port .....	41
11 Accessory mounts .....	43
12 IMU data .....	44
12.1 Accelerometer .....	44
12.2 Gyroscope .....	44
12.3 Magnetometer .....	44
12.4 Coordinate system for IMU .....	44
Appendix A File structure on the SD card .....	46
A1 Example recording.g3 file .....	46
A2 Recordings folder .....	47
A2.1 Scene Camera Object .....	47
A2.2 Eye Camera Object .....	48
A2.3 Gaze Object .....	48
A2.4 Camera Calibration Object .....	48
A2.5 Snapshot Object .....	48
A2.6 Events Object .....	49
A2.7 IMU Obejct .....	49
A2.8 Meta folder .....	49
A3 On disk data format .....	49
A3.1 Gaze data .....	49
A3.2 Sync Port Event Data .....	51
A3.3 Event Data .....	52
A3.4 IMU Data .....	52

# 1 Introduction

This document is intended for readers who want to integrate Tobii Pro Glasses 3 into their own software or research processes, either using the live API to control the glasses hardware, or in the analysis phase, to process data that is captured by the glasses.

Readers who plan to integrate using the Live API are assumed to have software development knowledge and understand concepts like REST API, asynchronous calls, JSON objects, etc.

Tobii Pro Glasses 3 has an API that can be accessed from most modern programming languages. For this reason, this document will be language agnostic and only explain the concepts and workflows needed when developing software with the glasses. There is a reference implementation of most workflows in JavaScript/HTML in the Tobii Pro Glasses 3 Example web client.

## 1.1 Firmware version

This version of the Tobii Pro Glasses 3 Developer Guide is updated to and including firmware version 1.33.

## 1.2 Units and definitions

The following units and definitions are used throughout the API:

- **Timestamps** are always in seconds with decimals if nothing else is indicated.
- **Gaze2D** is in normalized video coordinates, (0,0) corresponds to the top left corner of the scene camera video and (1,1) corresponds to the bottom right corner.
- **Gaze3D** is a position in millimeters from the scene camera, X is positive to the left, Y is positive up, and Z is positive forward.
- **Gaze Origin** is a position in the same coordinate system as Gaze3D.
- **Gaze Direction** is a normalized vector that starts in the gaze origin of the respective eye, same coordinate system as Gaze3D.
- **Pupil diameter** is measured in millimeters.
- **Accelerometer** is measured in meter/second<sup>2</sup> (m/s<sup>2</sup>).
- **Gyroscope** is measured in degrees/second (°/s).
- **Magnetometer** is measured in microtesla (μT).

## 1.3 Document conventions

In this document, the network address of Tobii Pro Glasses 3 will be written as "<g3-address>". You can replace this with an IPv6 address, an IPv4 address, the hostname (<serialnumber>) or the mDNS address record of your recording unit (<serialnumber>.local).

## 1.4 Tobii Pro Glasses 3 Example web client

Tobii Pro Glasses 3 has a built in web interface called the Example web client. This web interface provides both a HTML/JavaScript reference implementation of all of the common workflows, and

a complete and interactive API reference documentation. Point your web browser to the address of your Tobii Pro Glasses 3 (<http://<g3-address>>).

## 1.5 Tobii Pro Glasses 3 .net SDK

There is a .net client library for Tobii Pro Glasses 3 that wraps the API in convenient .net classes. It supports both .net framework and .net core/.net5. You can install the library from nuget. <https://www.nuget.org/packages/Tobii.Glasses3.SDK>

The source is available on GitHub. The GitHub repository also has some demo applications. <https://github.com/tobiipro/Tobii.Glasses3.SDK>

## 1.6 Tobii Pro Glasses 3 python SDK

There is also a python client library for Pro Glasses 3. It supports discovery and communication over websocket for signals, properties, and actions. It is built with asyncio for all network calls. It has native python RTSP support with both transport (via aiortsp), demuxing, and decoding (via ffmpeg). There is a demo application built with the kivy UI framework.

All source is available here:

<https://github.com/tobiipro/g3pylib>

## 2 License agreement

Usage of the Tobii Pro Glasses 3 API is subject to the Terms and Conditions of Tobii Pro Software Development License Agreement for Research Use. For more information, go to [Tobii Software Development License Agreement](#). A copy of the license agreement is also available in the Glasses 3 Example web client.

## 3 API Concepts

Tobii Pro Glasses 3 has a REST API that allows you to communicate with the recording unit to start/stop recordings, perform personal calibrations, get live data streams, etc. that can be accessed either through HTTP requests or WebSocket communication.

### 3.1 WebSocket

The WebSocket connection is based on sending messages with JSON objects across the socket. Since WebSocket offers asynchronous communication (as opposed to request/response in HTTP communication) each request needs to contain a unique id that will be included in the response message that is sent back over the WebSocket.

The address to connect the WebSocket is `ws://<g3-address>/websocket`, and you must specify the sub protocol “`g3api`” when you connect. The sub protocol is specified by setting the HTTP request header “`Sec-WebSocket-Protocol`” to “`g3api`” when setting up the WebSocket.



**Special note for .net developers:** Due to an incorrect HTTP header “`Content-length:0`” from Tobii Pro Glasses 3 in one of the hand-shake messages when creating the WebSocket, the .net `ServicePoint` will immediately schedule the WebSocket for cleanup. The default timeout for this cleanup is 100 seconds. Once cleaned, the WebSocket will no longer receive or send any messages. Until this incorrect header has been removed, the easiest workaround, is to change the default idle time to a high value:

```
ServicePointManager.MaxServicePointIdleTime = int.MaxValue;
```

### 3.2 HTTP

To make an API request over HTTP, you combine the root URL of the API with the path of the API element you want:

```
http://<g3-address>/rest/<path-to-api-element>
```

Tobii Pro Glasses 3 does not support secure/encrypted HTTP (HTTPS).

### 3.3 Objects

All functionality in the API is grouped into objects. The available objects are listed as children of the root object in the API and include:

- `calibrate`- functionality for making user calibrations
- `neighborhood`- functionality for detecting other glasses on the network using Zeroconf
- `network`- configuration for network settings, both LAN and wireless network settings
- `recorder`- functionality for making recordings



- `recordings`- lists the recordings that are available on the SD card and their metadata
- `rudimentary`- simple interface for accessing data streams and scene camera images during development, testing and debugging
- `settings`- persistent configuration
- `system`- system information such as serial number, system time/time zone. Child objects for accessing battery and SD-card information
- `upgrade`- functionality related to firmware upgrade
- `webrtc`- live view video and gaze data over WebRTC

For a full overview of all objects and their usage, please refer to the reference implementation and API reference documentation in the Tobii Pro Glasses 3 Example web client. For more information, read [Tobii Pro Glasses 3 Example web client](#).

## 3.4 Properties

Properties are read only or read/write values in the API. Each property belongs to an object and is accessed with the path of the object followed by a period "." and the property name.

**Syntax:** <path to parent object>.<property name>

**Example:** `system.recording-unit-serial`

### 3.4.1 Read a property value

To read a property, use a GET-request with the path of the property. The response will contain the value of the property.

Example HTTP:

HTTP GET	<code>http://&lt;g3-address&gt;/rest/system.recording-unit-serial</code>
Response	<code>"TG03B-080200004381"</code>

Example WebSocket:

Send	<code>{ "path":"system.recording-unit-serial", "id":22, "method":"GET" }</code>
Receive	<code>{ "id":22, "body":"TG03B-080200004381" }</code>

### 3.4.2 Write a property value

To write a new value to a read/write property, use a POST request with the new value as the body. The response when setting a property value is a boolean that indicates if the setting of the value was successful or not.

Example writing a property with HTTP request:

HTTP POST	<code>http://&lt;g3-address&gt;/rest/recorder.visible-name</code>
Body	<code>"Rec1"</code>
Response	<code>true</code>

Example of writing a property with WebSocket connection:

Send	{ "path": "recorder.visible-name", "id": 39, "method": "POST", "body": "Rec1" }
Receive	{ "id": 39, "body": true }

### 3.5 Actions

Actions are used to execute commands through the API. Examples are starting and stopping a recording, deleting a recording, creating a WebRTC-session. Just like properties, actions belong to objects.

**Syntax:** <path to parent object>!<action>

**Example:** recorder!start

When calling an action, you must always provide the arguments as the body of your request. The body should be a JSON array. If there are no arguments, the body should be an empty array ([]). Failure to send a valid body will result in an error.

The result of the action is returned as a JSON value/object. For HTTP requests, it will be the entire response, for WebSocket requests, it will be the body property of the matching response object.

Execute an action with HTTP request example:

HTTP POST	http://<g3-address>/rest/recorder!start
Body	[]
Response	true

Execute an action with WebSocket connection example:

Send	{ "path": "recorder!start", "id": 17, "method": "POST", "body": [] }
Receive	{ "id": 17, "body": true }

Execute an action with parameters using a HTTP request example:

HTTP POST	http://<g3-address>/rest/recorder!meta-lookup
Body	["MyKey"]
Response	"c2FtcGx1IGNvb3R1bnRzCg=="

Execute an action with parameters using WebSocket example:

Send	{ "path": "recorder!meta-lookup", "id": 64, "method": "POST", "body": ["MyKey"] }
Receive	{ "id": 64, "body": "c2FtcGx1IGNvbnRlbnRzCg==" }

"c2FtcGx1IGNvbnRlbnRzCg==" is the Base64 encoded representation of the string "sample contents"

If an action fails, it can provide additional information about the failure in an error-info property in the response. This is an example of trying to start a recording without an SD-card inserted:

HTTP request:

HTTP POST	http://<g3-address>/rest/recorder!start
Body	[ ]
Response	False
Response headers	X-g3-action-error = resource;/system/storage

WebSocket request:

Send	{ "path": "recorder!start", "id": 17, "method": "POST", "body": [] }
Receive	{ "id": 17, "body": false, "error-info": {"type": "resource", "path": "/system/storage"} }

## 3.6 Signals

Signals allow for data to be pushed asynchronously from the glasses to the API client. While it is possible to subscribe to signals using HTTP requests, it is not advised since it will only allow a single message per subscription. The risk of losing messages before the client has placed a new subscription is significant. To subscribe to a signal using WebSocket, send a message that contains the path and a unique id. The response will contain the same id and the body will contain the identity that each data package in the signal will contain.

Send	{ "path":"system/storage:state-changed", "id":28, "method":"POST", "body":null }
Receive	{ "id":28, "body":454 }

This response tells us that future data packages that are pushed as a part of the `storage:state-changed` signal will have the signal id 454. If the SD card is removed, the following messages will be sent by the RU:

Receive	{ "signal":454, "body":["good", "not-available"] }
Receive	{ "signal":454, "body":["unknown", "not-inserted"] }

The values in the body comes from the API properties `storage!space-state` (with values "good", "low", "verylow" and "unknown") and `storage!card-state` ("not-inserted", "not-available", "read-only", "available", and "failure"). If any of these properties changes, the signal will send a notification. Other signals like a `gaze-signal` will send new values much more frequently.

All signals will have a body that is an array of values in order to send multiple values in a notification. This array behaves more like a record than an array. For a given signal, the number of values will always be the same. In the example above, the signal `system/storage:state-changed` sends two values in order to inform about both the `space-state` and the `card-state`. The signal `recordings:child-added` sends a notification each time a recording has been added. When a new SD-card is inserted, multiple recordings might be added, and the signal will notify once per recording, with exactly one recording id in each notification.

### 3.6.1 HTTP vs. WebSocket

All functionality in the API can be used over either HTTP requests or WebSocket requests. Using WebSockets has the following advantages:

- Signals can support multiple responses
- Less resource intensive

## 4 Workflows

### 4.1 Discovery

Discovery is the process of finding the network address for a Tobii Pro Glasses 3 unit once it is connected to the same network as the API client.

#### 4.1.1 Zeroconf

Tobii Pro Glasses 3 can be located using Zeroconf. The device exposes the following services:

<b>Service:</b>	<b>_tobii-g3api._tcp.</b>
Address	<ip6-address>
Address	<ip4-address>
Port	80
<b>Text properties:</b>	
path	/rest/

REST API clients should use the path property of the Zeroconf service announcement to build the base path of the API. If the path to the REST API changes in future firmware releases, the Zeroconf service announcement will provide the correct value.

<b>Service:</b>	<b>_rtsp._tcp</b>
Host	<serialnumber>.local
Address	<ip6-address>
Address	<ip4-address>
Port	8554
<b>Text properties:</b>	
path	/live/all
Recordings	/recordings

#### 4.1.2 Hostname

The device will have the same hostname as its own serial number. This hostname will be exposed to external DNS servers, so if the DNS server on the network that the glasses is connected to accepts hostnames from clients, this name can be used to access the API.

#### 4.1.3 mDNS

Tobii Pro Glasses 3 will register itself on any network with a unique hostname based on the serial number, and it will present the mDNS-name <serialnumber>.local (i.e. TG03B-080200004381.local). This name can be used to access the REST API (<http://TG03B-080200004381.local/rest>) or to set up a WebSocket connection (<ws://TG03B-080200004381.local>).

#### 4.1.4 Wireless access point

The default network setting of Tobii Pro Glasses 3 starts a wireless point that any client can connect to. When connected to the default Tobii Pro Glasses 3 access point, the device itself will always have the same IPv4 address (192.168.75.51).

SSID	<serialnumber>
Password	TobiiGlasses
Subnet Mask	255.255.255.0
Default gateway	192.168.75.51
DNS server	192.168.75.51
DHCP server	192.168.75.51
DHCP Lease time	24h
NTP	Not available
Frequency	2.4 GHz
Channel	1

When configured as an access point the network will provide a DNS-server with a DNS record for the hostname (serial number).

The bandwidth available through the wireless network is ~20mbit/s. This means that it will not be possible to stream live video to multiple clients without significant risk of packet/frame loss.

If you need different settings for the access point it is possible to set up a custom Wi-Fi network configuration with other values than the above.

## 4.2 Live view

Tobii Pro Glasses 3 supports live view of the scene camera video over either WebRTC or RTSP. Both protocols offer relatively low latency (~0.2s) for the video and access to both scene camera video, eye camera video (4 camera images per video frame), and audio stream. The scene camera video stream has no gaze overlay by default. For more details on how to enable automatic gaze overlay in the video stream, read [Gaze overlay](#).

### 4.2.1 WebRTC

WebRTC is a video streaming protocol designed for use in a browser environment. WebRTC is supported in the current versions of all major browsers (Chrome, Firefox, Safari, Opera, Edge) and is the same protocol that is used in many video conference / video calling solutions.

The WebRTC implementation in Tobii Pro Glasses 3 is verified to be compatible with Chrome and Firefox on Windows, Linux, and macOS.



WebRTC is designed to handle connection of clients over the internet and through various network topologies with firewalls and NAT routers, Tobii Pro Glasses 3 currently only supports setting up a WebRTC session over a local network using the REST API.

### Setup WebRTC

The setup sequence looks like this:

Operation:	Client code	Tobii Pro Glasses 3 response
Create RTCPeerConnection in the browser	<b>JS:</b> <code>this.peer = new RTCPeerConnection(rtc_configuration);</code>	

Operation:	Client code	Tobii Pro Glasses 3 response
Subscribe to local ICE candidates	<b>JS:</b> <code>this.peer.onicecandidate = (ev) =&gt; this.new_local_ice(ev);</code>	
Subscribe to peer tracks	<b>JS:</b> <code>this.peer.ontrack = (ev) =&gt; this._peer_track(ev);</code>	
Create the WebRTC session on Tobii Pro Glasses 3	<b>API:</b> <code>var uuid = webrtc!create([])</code>	uuid of WebRTC session
Subscribe to remote ICE candidates	<b>API:</b> <code>webrtc/&lt;uuid&gt;!new-ice-candidate</code>	
Request remote offer	<b>API:</b> <code>var offer = webrtc/&lt;uuid&gt;!setup([])</code>	SDP offer
Send remote offer to browser	<b>JS:</b> <code>this.peer.setRemoteDescription(new RTCSessionDescription({ type:"offer", sdp:offer.to_sdp() }));</code>	
Send local ice candidates to RU	<b>API:</b> <code>webrtc/&lt;uuid&gt;!add-ice-candidate([index, candidate])</code>	
Add remote ice candidates to browser	<b>JS:</b> <code>this.peer.addIceCandidate(candidate)</code>	
Create local answer	<b>JS:</b> <code>var answer = this.peer.createAnswer();</code>	
Send the local answer to the RU	<b>API:</b> <code>webrtc/&lt;uuid&gt;!start([answer])</code>	
Hook up the peer track to a video element		

Once the live view is active, it is important to let the glasses know you are still listening by calling the `webrtc/<uuid>!keep-alive` action on a regular basis. This call should be executed at least once every 5 seconds. If the keep-alive method has not been called for 20 seconds, the WebRTC session will assume that the client has lost the connection and terminate. In this case, the `webrtcsession:timed-out` signal will send an event that indicates this.

There is a reference implementation of setting up a live WebRTC stream in the Example web client.

### WebRTC data channels (added in FW 1.11)

When a working WebRTC connection has been established, there will be data channels available for receiving gaze data, events, IMU data, and syncport events.

## Additional information about WebRTC

Chrome and other browsers try to avoid exposing local IP-addresses to the JavaScript environment, and will therefore place anonymous mDNS addresses in the ICE candidates:

```
candidate:3178462403 1 udp 2113937151 b1feef5-febf-4714-b1ee-7a2b35cf-dfcc.local 52432 typ host generation 0 ufrag jhdo network-cost 999
```

These will not always be resolved correctly by the WebRTC server on the glasses. There is an API action `!remote-host` that can be used to query the glasses for your own IP-address and then replace any anonymous address in the contents of the ICE candidate with your explicit address. This is exactly what the “Resolve Zeroconf in Browser” checkbox in the Example web client Live View tab does.

Browsers do not always include link-local IPv6-addresses in the ICE candidates. On a direct ethernet cable connection it is more robust to use link-local IPv4 addresses when connecting to the glasses.

### 4.2.2 RTSP

RTSP is a video streaming protocol that doesn't require as much setup as WebRTC but is primarily suited for native applications rather than web applications. To start the live RTSP stream of the Tobii Pro Glasses 3 system, just connect an RTSP client to `rtsp://<g3-address>:8554/live/all`.

RTSP Streams:

Payload	Type	Content	Details
96	Media	Scene camera video	h264, 25fps, 1920x1080px
97	Media	Microphone audio	mpeg3, 24khz, 32bits/sample, mono
98	Media	Eye camera video	h264, 50fps, 1024x256px
99	Data	Gaze data	
100	Data	Sync in/out	
101	Data	IMU data	
102	Data	Events	

The media streams have been tested with VLC, an open-source video player available for multiple operating systems including Windows, macOS X, Linux, Android, and iOS. To open a live video stream in VLC, go to the main menu and select “Media” and then “Open network stream”.

### 4.2.3 Render gaze in live view

The scene camera stream does not have any gaze overlay in it by default, only the images that come from the scene camera sensor. This allows a client to choose the look and feel of the rendered gaze marker.

In both WebRTC and RTSP, there are additional data channels that provide live gaze samples. The samples contain complete eye tracking information (2D gaze, 3D gaze, gaze origin, gaze direction and pupil diameter). The data for 2D gaze will be in normalized video coordinates and can be used to visualize a marker on the video that represents the gaze position.



In live view it is often preferable to perform a bit of noise reduction on the gaze position before overlaying it on the scene camera image.

It is not possible to get exact sync of gaze/video in WebRTC. Rendering the latest gaze sample is currently the best option. Sync information will be provided in future FW updates.

The data streams in RTSP have timestamps that allow synchronization with the media streams.

RTSP clients also has the possibility to request gaze overlay embedded in the video stream by adding a parameter to the RTSP URL:

```
rtsp://<g3-address>:8554/live/all?gaze-overlay=true
```

For more information about gaze overlay, see [Settings](#).

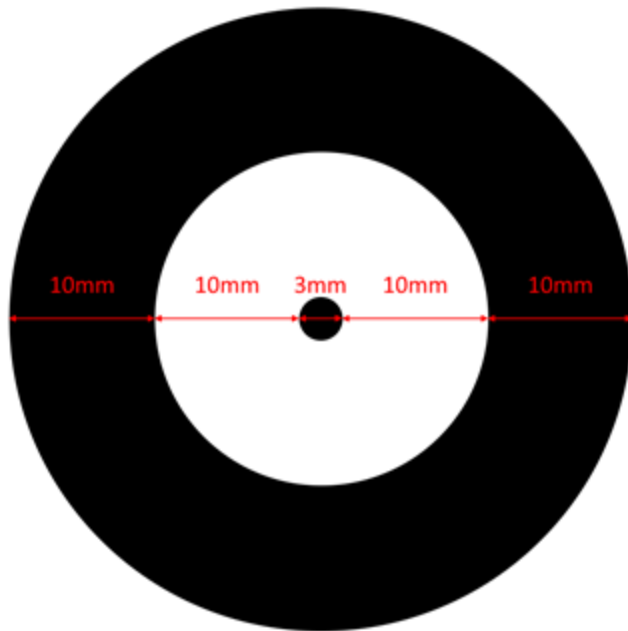
## 4.3 Calibration

It is possible to use the eye tracking in the glasses without calibration, but the accuracy error will be significant. To get better eye tracking accuracy, the eye tracker needs to be calibrated for the person wearing the glasses. Tobii Pro Glasses 3 supports two methods for calibration. Calibration with a calibration target requires that the user looks at the target and is the method that gives the best accuracy. Calibration without a calibration target (auto calibration) can be done without instructing the user to look at any specific target. For some persons this will give good results, for others, there will be a significantly larger error.

### 4.3.1 Calibration with calibration target

Tobii Pro Glasses 3 comes with preprinted calibration targets for this purpose. The targets should be positioned in front of the person at around 50-100 cm and held still before the calibration is performed.

The diameters of the circles in the marker are 43mm/23mm/3mm. If a target with the wrong dimensions is used, the estimated distance to the target will be wrong and the eye tracking error will increase. Exact color is less important, but the contrast between the circles and the center dot must be high, and the outer circle and center dot must be darker than the middle circle.



During the calibration workflow, it is advisable to enable the marker stream and render the marker on the live video to make it easy to verify that the calibration target is correctly identified. This is done by calling the action `calibrate!emit-markers`. It will enable the detection for 3 seconds or until a calibration is completed. While the marker detection is enabled, the signal `calibrate:marker` will emit values for each video frame that has been processed.

The body of the marker stream will be an array of values that looks like this:

```
[1.869804, [-210.64152, 72.88064, 686.21405], [0.64479, 0.40370]]
```

The first value is a timestamp. The second value is the 3D coordinate of the marker (x, y, z) in millimeters from the scene camera. The third value is the 2D coordinate of the marker in normalized video coordinates.

When the marker is correctly identified in the scene camera stream, and the participant is focusing on the calibration target, call the `calibrate!calibrate(true)` action to perform the calibration. If the calibration is successful, the result will be “success” and the gaze marker overlay should “snap” to the calibration target in the scene camera video.

If the calibration fails for some reason, the result of the action will be “fail-” followed by the reason for the failure:

These are the failure reasons that can occur:

<code>fail-gaze-inconsistent</code>	Happens if the user moves the eyes (or the head) during the calibration (and can only happen for calibration with target where stable gaze is important).
<code>fail-eyes-not-detected</code>	Eyes can't be found in the eye images. For the calibration to work, both eyes needs to be fully visible in both eye cameras.

fail-marker-not-detected	<p>This indicates that the target can't be found in the scene camera image, there are several possible reasons for this:</p> <ul style="list-style-type: none"> <li>• Contrast of the target too low</li> <li>• Too dark or too bright to show the contrasts of the target in the scene camera image</li> <li>• Reflections in the target</li> <li>• Incorrect proportions of the target</li> <li>• Target too close or far away</li> <li>• Other items in field of view that resembles the calibration target</li> </ul>
fail-parameters-out-of-bounds	<p>Happens if the user is not looking at the target, or the target detection accidentally finds something other than the target. The error message really means that it was possible to do a calibration, but that the calibration parameters are so out of range that they can't reasonably represent a human eye.</p>

Detecting the calibration target in the scene camera video stream is a rather resource/CPU intensive operation and should not be done during recording of eye tracking data as it can negatively impact the possibility to calculate correct gaze data.

The previous call `calibrate!run()` is still supported and is equivalent to the new method `calibrate!calibrate(true)` but has a less detailed response (only true/false).

There is no requirement to subscribe to gaze data or have an active live view during the calibration, but it will make it easier to verify that all is correctly setup before the calibration is performed. Keep in mind that gaze coordinates will be incorrect until a good calibration has been performed. If Tobii Pro Glasses 3 is just started, the gaze will be using the default calibration. Once a calibration has been performed, the calibration will be active until a new calibration is performed and will not be affected by starting or stopping recordings. There is currently no action to reset the calibration.

#### 4.3.2 Calibration without calibration target

Firmware 1.33 introduced the possibility to do a calibration without a target. This calibration can be done as long as the glasses are worn, and does not require the participant to be focusing on anything in particular. It is preferable if the gaze is in the center region of the field of view, but that is where gaze is a large portion of the time anyway. The resulting accuracy will not be as good as when doing a calibration with a target, but the workflow is easier and it can be done even in low light conditions where the low contrast of the target can make it hard to detect during calibration.

To perform a calibration without a target, call `calibrate!calibrate(false)`. The possible result codes are a subset of the result codes when calibrating with a target. Only “success”, “fail-eyes-not-detected” and “fail-parameters-out-of-bounds” are possible.

## 4.4 Make a recording

The normal workflow when making a recording is:

- Make a successful calibration
- Validate in the live view that the gaze is correctly positioned on the object that the user is looking at
- Start the recording
- Set the name of the recording and other metadata
- Stop the recording

Between starting and stopping the recording, all data will be stored on the SD card and there is no requirement to have an active live view or even a network connection with a client. If the network connection between the client and the glasses is lost, the recording will continue until a client reconnects and stops or cancels the recording. If no client connects, the recording will continue until Pro Glasses 3 runs out of battery or space on the SD card.

A controlled power down (pressing the power button for 4 seconds, releasing it when the LEDs change from blinking to solid yellow and then wait until the LEDs turn off) will also terminate the recording. Removing the battery, SD card, or performing a forced power down (pressing the power button for 12 seconds) will in most cases render the recording corrupt.

#### 4.4.1 Start, stop, cancel

The `recorder!start` action will start storing data on the SD card (scene camera video, eye tracking data, events, metadata, and IMU data). The start action can fail for a number of different reasons:

- No SD card
  - `("type":"resource", path:"/system/storage")`
- Not enough space on the SD card
- SD card is read only
  - `(type:"resource",path:"/system/storage")`
- Not enough battery power
- No head unit connected
  - `(type:"internal",path:"/recorder")`
- Already a recording in progress
  - `(type:"not-permitted",path:"/recorder")`
- There is a folder on the SD card called "recorder"
  - `("type":"internal")`

Once the start action has succeeded, the recording will be in progress. At this point the properties of the recorder-object will provide useful information about the ongoing recording:

- `recorder.created` - the UTC time when the recording was started.
- `recorder.current-gaze-frequency` - For the 100 Hz edition of Tobii Pro Glasses 3, this property will indicate if the glasses are currently operating at 100 Hz or 50 Hz. For the 50 Hz edition, this value will always be 50.
- `recorder.duration` - the number of seconds the recording has been in progress.

- `recorder.folder` - the name of the folder that the recording will be stored in once it is completed. This value can be set to override the default behavior. The character set that can be used is limited to what is allowed for file/folder names.
- `recorder.gaze-samples` - the total number of gaze samples that have been recorded, regardless of if gaze was detected or not.
- `recorder.valid-gaze-samples` - the number of successfully eye tracked samples. This can be used together with the gaze-samples property to indicate if there is a loss of tracking without having to subscribe to the gaze signal.
- `recorder.remaining-time` - the time until the recording will be force-stopped. This value will be calculated based on remaining battery, remaining space on the SD card and file system type of the SD card.
- `recorder.timezone` - the timezone that the glasses were configured with at the start of the recording.
- `recorder.uuid` - the unique identifier that can be used to address the recording after it has been stopped.
- `recorder.visible-name` - this property can be used to give the recording a human readable representation.

During the recording, it is possible to set metadata, folder, and visible-name as well as request the saving of snapshots from the scene camera. The data will be stored in a folder on the SD card called “recorder”.

Calling `recorder!stop` will stop the recording and finalize all files stored on the SD card.

The stop operation can fail for the following reasons:

- Invalid folder name
  - Folder-name too long
  - Folder-name contains invalid characters
  - Folder-name is “recorder”



Note, that trying to stop a recording when there is no recording in progress will return true.

After the recording has been stopped, the folder with all data from the recording will be renamed from “recorder” to the value of the `folder-name` property. The recording can be accessed via the API with the UUID that was assigned by the recorder object.

Canceling the recording will remove all data stored during the session. It is semantically equivalent to stopping and deleting the recording, but slightly faster.

#### 4.4.2 Visible-name vs. folder

The `recorder.folder` property will be used to create a folder on a FAT32/exFAT file system and is restricted in length and in which characters are allowed. The following characters cannot be used: `0x00-0x1F 0x7F " * / : < > ? \ |`. The `visible-name` property on the other hand can contain any Unicode character.

Default value for both the `folder` and `visible-name` is the UTC time when the recording was started, e.g. `20200710T100603Z`

Visible-name property is suitable to use as a human readable representation of the recording.

When there is no recording in progress, the `folder` and `visible-name` will both be null.

When the recording is started, the `folder` will assume its default value (see above), and `visible-name` will still be null. When the recording is stopped, if the `visible-name` is still null, the property will get the same value as the `folder` (if the `folder` has been changed from the default value, the `visible-name` will get the updated folder name).

Both `folder` and `visible-name` can only be set in the `recorder-object` while the recording is in progress. Once the recording is stopped, the `visible-name` is a read/write property in the `recording-object` that is created. The `folder` can be changed by using the `move-action` on the `recording-object`.

#### 4.4.3 Metadata in recordings

An API client can store arbitrary metadata in a recording using the metadata actions available in the `recorder-object` and the `recording-objects`. The metadata API consists of the following actions:

`meta-keys()`: <array of keys>

`meta-insert(key: string, data: <Base64 encoded data>): boolean`

`meta-lookup(key: string): <Base64 encoded data>`

The data that is stored using the `meta-insert` action will be stored in a subfolder of the recording folder on the SD card. The file name will be the same as the metadata key (so file name limitations apply to the name of the key). The key can contain period (".") to make the resulting file have a file extension. Using Base64 encoding means that you can store both binary and text data.

The metadata actions can be used to store additional recording information such as information about the participant or project.

When a recording is stopped, you can no longer use the metadata actions in the `recorder-object` to read/write metadata, but you can do it with an identical set of actions on the `recording-object`.

#### 4.4.4 Recording events during a recording (FW 1.10+)

During the recording, clients can send client specific events through the API and get them timestamped and stored as a part of the recording.

A custom event consists of a tag and a JSON object. The tag should indicate what the event represents or how it should be interpreted. The tag could for example be the name of the data source, or the type of data it contains.

To send a custom event to the API, use the action `recorder.send-event ([<tag>, <object>])`.

Example of sending a custom event using a HTTP request:

HTTP POST	<code>http://&lt;g3-address&gt;/rest/recorder!send-event</code>
-----------	---

Body	<pre>[   "gps-location",   {     "lat":59.400539,     "long":18.038402   } ]</pre>
Response	true

### Example of sending a custom event over WebSocket

Send	<pre>{   "path":"recorder!send-event",   "id":99,   "method":"POST",   "body":[     "gps-location",     {       "lat":59.400539,       "long":18.038402     }   ] }</pre>
Receive	<pre>{   "id":99,   "body":true }</pre>

The GPS location can change continuously during the recording, and each time the event is sent to the API, it will be timestamped and stored. It will also be sent to any subscriber of the `event-signal` (available in WebRTC session objects and the rudimentary object).

The signal would look like this assuming that the signal was assigned signal id 930 when the subscription was created (the values that comes from the `send-event` action in bold font):

```
{
  "signal":930,
  "body":[
    81.410,
    {
      "tag":"gps-location",
      "object":{
        "lat":59.400539,
        "long":18.038402
      }
    }
  ]
}
```

Keep in mind that events recorded using the event API are immutable and cannot be modified through the API.

It is possible to use the metadata API to store information that needs to be mutable during the recording and/or after the recording is complete. Tobii Glasses Controller stores user events using the metadata API using the following structure:

```
{
  "id": "user-event-81bcc49498ff.json",
  "application": "controller",
  "applicationVersion": "1.17.8",
  "timestamp": 9.782,
  "label": "<custom event info>",
  "color": "#FF9F0A"
}
```

This event format is also supported by Tobii Pro Lab when the recording is imported for analysis.

#### 4.4.5 Take snapshots during the recording

Snapshots are JPEG images taken from the scene camera that are stored on the SD card. By default, a single image is saved in the beginning of the recording. Using the `recorder!snapshot` action, additional snapshots can be taken during the recording. They will be stored in the recording folder and the timestamp of each snapshot is available in the recording.g3 file. Snapshots are stored with full resolution. No additional EXIF-metadata is stored in the image file.

#### 4.4.6 Duration of the recording

During a recording, the client can query the `recorder.duration` property to know how long the recording has been in progress. When there is no recording in progress the value will be -1. When the recording is started, the value is undefined until the `recorder!start`-action has completed. The `duration`-property does not report the time since the recording was started, but rather how much data has been flushed to disk. This means that the value is monotonous (steadily increasing), but not linear. The client can extrapolate (dead reckoning) using its own clock to get a good and smooth estimate of the recording duration, and only query the API occasionally to compensate for any clock drift between the glasses clock and the client clock.

### 4.5 Replay

The recording data stored on the SD card can be accessed in several different ways.

#### 4.5.1 Access recording data over HTTP

To access the recording data over HTTP, you first need to get the http-path for the recording. This is done using the API property `recordings/<uuid>.http-path`.



Request	<code>http://&lt;g3-address&gt;/rest/recordings/04c097a3-eaf1-454f-b541-488920974c0e.http-path</code>
Response	<code>/recordings/04c097a3-eaf1-454f-b541-488920974c0e</code>

Combine the base path with the address of the glasses `http://<g3-address>/<http-path>`.

Example:

`http://<g3-address>/recordings/0b5aaf39-1407-4a2b-825c-a12914ab1783`

This request will return the contents of the file recording.g3 in JSON format.

A description of the recording.g3 file is available in the appendix, [File structure on the SD card](#).

This JSON-object will contain information about the related files such as scene camera video and gaze data file (currently scenevideo.mp4 and gazedata.gz). The file names may change in the future, so it is advised to read the actual file name using the properties in the JSON file.

`http://<g3-address>/recordings/<uuid>/scenevideo.mp4`

`http://<g3-address>/recordings/<uuid>/gazedata.gz`

Adding the URL parameter `use-content-encoding=true` will add headers to the response that will make a browser decode the gzip compression on the fly and allow you to treat the file as a text file.

`http://<g3-address>/recordings/<uuid>/gazedata.gz?use-content-encoding=true`

The content of the gaze file is a text file that has one JSON object per line (here formatted with line breaks for readability):

```
{ "type": "gaze", "timestamp": 681.751, "data": { "gaze2d": [0.535, 0.477], "gaze3d": [-32.499, -4.491, 345.359], "eyeleft": { "gazeorigin": [30.221, -11.798, -23.935], "gazedirection": [-0.160, 0.0432, 0.986], "pupildiameter": 2.653 }, "eyeright": { "gazeorigin": [-31.116, -12.392, -21.406], "gazedirection": [-0.011, -0.00270, 0.9999], "pupildiameter": 2.374 } } }
```

The timestamps of gaze data, events, and IMU data will correspond to the timestamps of the video.

In the Example web client, there is a reference implementation of showing recording information and replaying recordings with gaze overlay using the HTTP method in the tab called "Storage".

#### 4.5.2 Replay a recording with RTSP

The RTSP server on Tobii Pro Glasses 3 allows replaying a recording that is on the SD card.

`rtsp://<g3-address>:8554/recordings?uuid=<uuid>`

Replay via RTSP allows seeking in the video using the standard RTSP API.

Gaze data will be available as RTSP data streams.

Gaze will be visible in the video file only if the gaze-overlay feature was enabled during the recording. Gaze will not be added to the video during replay regardless of the value of the gaze-overlay setting.

#### 4.5.3 Replay a recording with WebRTC

WebRTC can be used to replay existing recordings as well. Create a WebRTC session using the action `webrtc!play(<uuid>)` and provide the `uuid` of the recording as a parameter for the action. For details on how to setup the WebRTC session, see the section about live view.

Replay via WebRTC does not currently support seeking in the video. Gaze data is available as WebRTC data channel.

In the Example web client, there is a reference implementation of showing recording information and replaying recordings with gaze overlay using the WebRTC method in the tab called "Recordings".

## 4.6 Time synchronization and NTP

When NTP (Network Time Protocol) is enabled through the API (`system.ntp-is-enabled`, default on) Tobii Pro Glasses 3 will regularly try to find an NTP-server if it is connected to an external network over LAN or Wi-Fi. If the recording unit (RU) is connected at boot time, it will usually sync immediately. If it has been running without a network, it can take significant time until the NTP-sync happens (30-60 minutes). Use `system.ntp-is-synchronized` to determine if the glasses are synchronized or not. If the RU is synced to an NTP-server and is subsequently disconnected from the network where the NTP-server is available, it will still consider itself synced for a time (8+hours). After reboot, the RU will no longer consider itself synced but the clock will still be relatively correct (the glasses system clock has a separate battery backup that lasts for several weeks).

The default NTP-server is `debian.pool.ntp.org`, but if the glasses are connected to a network with a DHCP that specifies a different NTP-server, it will be used instead.



Note that this only works if the glasses are connected to the network with a custom NTP-server when it is powered on. Switching between networks with custom NTP-servers will require a reboot of the glasses to take effect. There is no API to see which NTP-server is currently used.

To set the device clock through the API, the client must ensure that the NTP-functionality is disabled (regardless of whether the RU is synced or not), otherwise the attempt to set the time will fail. To do this, you set `system.ntp-is-enabled` to false. Setting the time zone will always be possible, regardless of the RU is NTP-synced or not.

When setting the system time, an explicit time zone needs to be specified. The format for setting the time to a UTC time looks like this:

- `yyyy-mm-ddThh:mm:ss[.dddddd]Z`
- **Example:** `2020-07-16T07:56:04.089725Z`

Note that the leading zeros are required. Z is the abbreviation of UTC time zone (aka the “Zulu time zone”). The decimal seconds are optional.

To set the system time using local time, an explicit “time zone designator” should be provided and should be specified as an offset from UTC. The equivalent of the above time with a time zone for Sweden/DST (2h ahead of UTC) would be:

- `2020-07-16T09:56:04.089725+02:00`

In most cases, it is easier to convert the local time to UTC before setting the time.

## 4.7 Time zone

Setting the time zone will have the following effects:

- `recorder.timezone` and `recordings/<uuid>.timezone` properties will indicate the time zone that was active when the recording was started
- Creation date/modified date of files on the SD card will be in the configured time zone

Setting the time zone does NOT affect:

- The default folder names of recordings
- The value of the `recorder.created-property`
- The value of the `recordings/<uuid>.created-property`
- The `system.time-property`

All of the above will always use UTC notation, regardless of the time zone setting.

## 4.8 Check status of the glasses

The API provides a number of ways to check the status of the device.

### 4.8.1 SD card status

The general status of the SD card is available in the property `system/storage.card-state`. This property is an enum with the following alternatives:

<code>not-inserted</code>	No SD card inserted in the SD card slot.
<code>not-available</code>	SD card is inserted but cannot be mounted. One reason could be that the SD card is not formatted, or has an unsupported file system.
<code>read-only</code>	SD card is read only. The SD card has a small switch on the side that prevents devices from writing to the SD card. An SD-card will also be remounted as read only if an error occurs during reading or writing to the card (to protect the existing data from further damage).
<code>available</code>	SD-card is inserted, no problems detected.
<code>failure</code>	SD-card is inserted, but the card has errors and cannot be used.

Removing the SD card while it is being read or written to is likely to cause data loss and is strongly advised against. There is no indication on the recording unit itself when the SD card is mounted, but there is a property in the API that indicates when the card should not be removed: `system/storage.busy`

The size and available space on the SD card can be read with the properties `system/storage.size` and `system/storage.free`. Both properties will report the value in bytes.

The property `system/storage.remaining-time` provides information on how much time can be recorded on the SD card. Since the size of the video file and other files in the recording varies depending on the contents, this estimate is not exact.

An ongoing recording will be stopped with some margin if the SD card is about to run out of available space.

Both the `busy`-property and the `card-state` property have a signal that indicates if the value has changed.

#### 4.8.2 Battery status

The charge of the battery can be retrieved from the `system/battery` object.

<code>level</code>	The charge of the battery in percent (0.0-1.0).
<code>charging</code>	Indicates if external power is plugged in or not. The battery will charge very slowly when the external power is plugged in. It is much more efficient to charge the battery with the accompanying battery charger.
<code>remaining-time</code>	This is an estimate (in seconds) of how long recordings can be made with the battery in the current state. The glasses use relatively little battery when idle (not eye tracking, not transmitting video or data) compared to when the glasses are recording. A fully charged battery is capable of recording for approximately 1 h 45 min.
<code>state</code>	<p>This is a rough estimate of time left of battery charge states:</p> <ul style="list-style-type: none"><li>• <b>full</b> - battery is fully charged</li><li>• <b>good</b> - more than 15 minutes recording time left</li><li>• <b>low</b> - less than 15 minutes recording time left</li><li>• <b>very low</b> - less than 5 minutes recording time left</li><li>• <b>unknown</b> - no battery is inserted</li></ul> <p>Changes in this property will be indicated by the signal <code>system/battery:state-changed</code></p>

#### 4.8.3 Firmware update status

To check whether a firmware upgrade is in progress, there is a boolean property `upgrade.in-progress`. If an upgrade is in progress, it is possible to track the progress of the upload/upgrade by subscribing to the signal `upgrade:progress`. During the upgrade, it will emit values continuously that shows the upload and upgrade progress with a value between 0 and 1. When the upgrade is completed, the `upgrade:completed` signal will emit once with a boolean value indicating if the upgrade was successful or not.

### 4.9 Settings

The `settings`-object contains persistent settings that can be set by the user to change the operation of the glasses.

#### 4.9.1 Gaze frequency

For the 100 Hz edition of Tobii Pro Glasses 3, it is possible to select between 50 Hz and 100 Hz eye tracking.

gaze-frequency	This property can be used to change between the available frequencies. The supported values are 50 Hz and 100 Hz. If the glasses don't support 100 Hz (as indicated by <code>system!available-gaze-frequencies</code> ), this property can still be set to 100, but the glasses will run in 50 Hz. NB! in Firmware 1.11, This was an enum with the values "default", "50 Hz" and "100 Hz"
----------------	--

Related API elements:

<code>system!available-gaze-frequencies</code>	This action returns an array of the available frequencies ([50] for the 50 Hz edition and [50, 100] for the 100 Hz edition).
<code>recorder.current-gaze-frequency</code>	This property shows what frequency the eye tracker is currently running at. It is only available if there is a recording in progress.

#### 4.9.2 Gaze overlay

The scene camera video does not contain any gaze marker by default. This has several advantages:

- No part of the field of view is obscured by the gaze marker if image processing is applied to the scene camera image.
- Custom filtering can be applied to the gaze signal before it is rendered on the scene camera image.
- Client has full control over the look and feel of the rendered gaze marker (size, appearance, and color).
- Gaze can be rendered at a higher frequency than the video frame rate.

It also has some drawbacks:

- Standard video players will not be able to play the recorded video with gaze without first processing the video with custom software and adding the gaze marker (causing degradation due to re-encoding of the video).
- Gaze will not be visible in standard RTSP/WebRTC receivers.

In scenarios where there is no need to perform image processing of the scene camera video, and there is no need to process the gaze signal before rendering the gaze on the video, the gaze can be rendered on the scene camera images directly in the glasses.

When the gaze is rendered on the video frame, the gaze position will be linearly interpolated between the two samples before/after the timestamp of each video frame. Meaning that a client that renders a gaze marker based on the coordinates of a single sample will never be in the exact same position as the marker rendered by the glasses.

If the built-in rendering is enabled, this can be used to verify that a client that renders gaze itself handles the syncing between video and gaze correctly.

The gaze-overlay property is a boolean property that can be modified at any time by any client, even during a recording.

All clients will get the same video stream. Meaning if a client requests gaze overlay, all other clients (and the recorded mp4-video file) will also get gaze overlay.

<code>gaze-overlay</code>	Enable gaze overlay on the scene camera video frames.
---------------------------	---

Related API elements:

<code>recorder.gaze-overlay</code>	If a recording is in progress, this property indicates if the setting for gaze overlay was enabled when the recording was started.
<code>recording.gaze-overlay</code>	This read only property indicates if the gaze overlay functionality was enabled when the recording was started.

## 4.10 Scene camera exposure (added in FW 1.29)

The glasses scene camera uses dynamic exposure control. By default it is configured to take the entire scene into account and try to balance the exposure to get the best average exposure. In some scenarios with a large variations in illumination, this can lead to some areas being completely over or under exposed. Unfortunately, the dynamic range of the scene camera doesn't allow all areas in such a scene to be nicely exposed, but it is possible to prioritize correct exposure in the the area of attention instead of the entire scene.

Examples of when the gaze spot meter functionality can be useful:

- Mobile phone studies in low lit indoor settings
- Studies in cars during a sunny day

To control the scene camera exposure, use the following settings in the scene camera object:

<code>system/scenecamera .autoexposure-change-speed</code>	<p>This property control how rapidly the exposure adjusts when the illumination changes.</p> <p>0: the exposure will adjust infinitely slow (not at all), and can be used to lock the exposure at the current setting.</p> <p>0.7: typically gives a smooth and relatively quick adjustment.</p> <p>1: the exposure will adjust as quickly as possible. This will usually give a rather flickering impression, especially when combined with the gaze spot mechanism.</p>
<code>system/scenecamera .autoexposure-gaze-spot-weight</code>	<p>This property determines how much the exposure control is influenced by the area around the gaze.</p> <p>0: all areas of the scene image has the same importance when calculating the ideal exposure.</p> <p>0.5: the area around the gaze will have the same importance as all the rest of the scene together.</p> <p>1: only the area around the gaze is considered when calculating the ideal exposure.</p>

<code>system/scenecamera .autoexposure-gaze- spot-size</code>	<p>This property determines how large the area around the gaze-marker is considered for gaze spot metering. It is specified as the radius of a circle around the gaze marker and is measured in visual degrees.</p> <p>As an example, a 7 cm wide mobile phone viewed at 36 cm represents a visual angle of ~11 degrees horizontally across the screen, so a gaze spot size with a radius of 5 degrees would cover the width of the phone and give optimal exposure of the phone during a mobile phone study.</p> <p>Setting the gaze spot size to 15 degrees will correspond to a circle with a 16cm radius at 60 cm or approximately the height of a 24" screen at 60cm.</p> <p>To calculate the visual angle of an object, you can use the following formula: <math>2 * \arctan(0.5 * \text{&lt;size&gt;} / \text{&lt;distance&gt;})</math></p>
---	--

## 4.11 Zoom

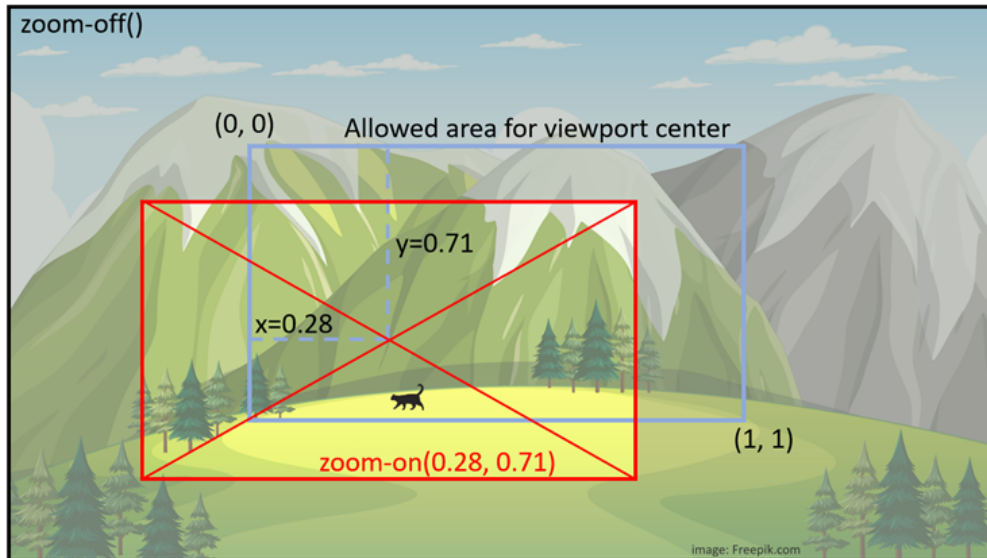
The scene camera in Tobii Pro Glasses 3 has a sensor resolution of 4K. Default mode is to scale the camera down to 1080p (1920x1080 pixels) but it can also be used to get higher level of detail at the cost of a smaller field of view.

*Zoomed out:* Scene camera sensor is scaled so that the original 4K image is binned to 1080p by merging 4 pixels into one. Field of view is 95 degrees horizontal and 63 degrees vertical.

*Zoomed in:* Scene camera sensor is cropped and the original pixels are preserved, but field of view is approximately half in both horizontal (48 degrees) and vertical (31 degrees).

In zoomed in mode, the viewport that is used can be adjusted to show any part of the zoomed out view. When switching to zoomed in mode, the center of the viewport needs to be specified as an x and a y-coordinate.

The x and y-coordinate represents the normalized coordinates of the center of the viewport, in a rectangle that is ¼ of the unzoomed area in an area.



2D gaze will always be mapped to normalized coordinates in the current viewport, regardless of whether it is zoomed in or zoomed out.

Gaze origin, gaze direction, 3D gaze and all IMU signals are unaffected by changes in zoom-area.

When the camera is in zoom-in mode, the area outside the zoomed viewport will not be available and will not be recorded.

It is possible to adjust the zoom field of view also when the camera is already in zoomed mode by calling the zoom-in action again. Calling it while the camera is still adjusting to the new region will not work, so it is advisable to wait 200 ms after each call before the action is called again.

<code>system/scenecamera.zoom-x</code>	Value between 0 and 1 representing the horizontal position of the zoom area if zoom is enabled.
<code>system/scenecamera.zoom-y</code>	Value between 0 and 1 representing the vertical position of the zoom area if zoom is enabled.
<code>system/scenecamera.zoomed</code>	Boolean property that is true if the camera is in zoom mode.
<code>system/scenecamera!zoom-off</code>	Disables zoom and restores full field of view.
<code>system/scenecamera!zoom-on(x, y)</code>	Enables zoom and will position the center of the zoomed area based on the x and y parameters. 0, 0 - zoomed area will be in the top left quadrant. 0.5, 0.5 - zoomed area will be in the center of the field of view. 1, 1 - zoomed area will be in the bottom right quadrant.

Scene camera exposure settings will work the same when the scene camera is zoomed in, but the exposure will be based on the visible pixels, not the zoomed out field of view.





**Zoom and Calibration:** Calibration with calibration marker can only be performed when the scene camera is not zoomed. If the camera is zoomed in when a calibration with marker is initiated, the camera will automatically be zoomed out. When the camera is zoomed in, the calibration marker stream will not report any detected markers either. It is recommended to zoom the camera out before starting the workflow of a calibration with marker.

## 5 SD card file system

Tobii Pro Glasses 3 supports SD cards formatted with either FAT32 or exFAT. The FAT32 file system is restricted to files smaller than 4GB (~1h 40min of G3 recording). exFat supports much larger files and is recommended for long recordings with an external power supply to the Tobii Pro Glasses 3 recording unit.

## 6 Streams and timestamps

Timestamp in all data streams in the Tobii Pro Glasses 3 API originate from the same physical clock, meaning that they will never drift relative to each other, but streams originating from different API objects will have different offsets to the internal clock. This means that a gaze sample in the gaze signal of a `webrtc` object will not have the same timestamp as the same sample in the gaze signal of the `rudimentary` object.

In the files in the recording folder on the SD card, all streams will begin from 0 (zero represents the first frame of the scene camera video).

Each data stream will always have a strictly increasing timestamp.

Each API object will use an offset that is calculated at the creation of the API object (this will be time 0).

## 7 Network configuration

Tobii Pro Glasses 3 is equipped with both an ethernet network port and a wireless adapter. The mechanism for configuring network settings is to create a network configuration object, set the properties of the new configuration, save it and finally activate it as the current configuration.

The Example web client has a convenient web page that simplifies manual creation of network configurations and shows the current status of the ethernet and wireless network adapters.

When configuring a network configuration, you must first select the address method for IPV4/IPV6.

The alternatives are:

<code>manual</code>	IP-address, gateway and name server must be configured explicitly. This method also supports enabling a DHCP-server on the glasses for setting up a small local network.
<code>dhcp</code>	IP-address, gateway and name server will be assigned by a DHCP server on the network.
<code>link-local</code>	IP-address will be assigned from the address range for link local addresses (for IP4 it will be 169.254.x.y, for IP6 it will start with fe80).
<code>disable/ignore</code>	The network interface will be disabled.
<code>automatic</code>	Network interface will try to find the most suitable configuration.

When configuring a network, it is possible to enable the interface to provide a DHCP-client.

The network configuration object exposes a subset of the properties available in the underlying network manager service. For more details on network configurations, please refer to <https://developer.gnome.org/NetworkManager/stable/>

### 7.1 Ethernet

For the ethernet adapter, there are two predefined network configurations:

<code>default</code>	Configuration for networks with a DHCP server. Both IP4 and IP6 will try to request a DHCP address. If there is no DHCP-server, the network adapter will be regularly restarted to get contact with a DHCP server.
<code>default-link-local</code>	Configuration for direct ethernet connection between the glasses and a client. Both IP4 and IP6 interfaces will be configured to assign link-local addresses (169.254.x.y / fe80::xxxx:yyyy:zzzz:vvvv).

If you need a custom configuration, you can create a new configuration and set all the network parameters through the API.

### 7.2 Wireless

To connect to an external wireless network, create a new network configuration using the `network/wifi!create-config` action. The return value of this action will be the UUID of the new network configuration. For most networks it is sufficient to configure the following properties:

- `network/wifi/configurations/<uuid>.ssid-name= ssid of your wireless network`
- `network/wifi/configurations/<uuid>.security= wpa-psk`
- `network/wifi/configurations/<uuid>.psk= password of your wireless network`

If the network is an open network, only the `ssid-name` property needs to be configured, the rest can be left with default values.

The `ssid-name` property accepts most commonly used SSIDs as a utf-8 encoded string. However, the SSID of a network is really a byte array and not a string and cannot always be represented as a string. The property `network/wifi/<uuid>.ssid` accepts a Base64 encoded version of the byte array that represents the SSID.

For more advanced networks or networks without DHCP capabilities, you might need to set additional properties for name servers, fixed IP-addresses etc.

When the required properties have been configured, the configuration should be saved using `network/wifi/configurations/<uuid>!save`. Finally the new configuration can be activated using `network/wifi!connect` with the UUID of the network configuration as the only parameter (Body = `["<uuid>"]`).

To reset the network settings to the factory defaults, call `network!reset`, this will remove all manually added network configurations for both ethernet and wifi. After resetting, the ethernet interface will support DHCP networks, and the wireless interface will provide an access point.

Example workflow to connect Tobii Pro Glasses 3 to a wireless network:

```
var uuid = network/wifi!create-config(["MyNetworkConfig"])
network/wifi/configurations/<uuid>.ssid <= "my-ssid"
network/wifi/configurations/<uuid>.security <= "wpa-psk"
network/wifi/configurations/<uuid>.psk <= "network password"
network/wifi/configurations/<uuid>!save
network/wifi!connect(["<uuid>"])
```

Alternative method by using the functionality to scan for networks:

```
network/wifi!disconnect // required to scan for available networks
network/wifi!scan
select an appropriate network definition from the child nodes of
network/wifi/networks
network/wifi!connect-network(
    ["<uuid of selected network>", "password"])
```

If there is no configuration with a matching SSID, a new configuration will be created and the SSID and the password will be saved with the configuration for future use. If there is already a configuration, the password will be reused if you do not provide a new password. A password override will not be saved to an existing configuration. To connect to an open network, there is no need to provide a password.

In access point mode, Tobii Pro Glasses 3 only supports 2.4 GHz network. When connecting to external networks (client mode) both 2.4 GHz and 5 GHz networks are supported.



With firmware prior to 1.33, only 2.4 GHz networks are supported also in client mode. 5 GHz networks will be detected when listing available networks, but connection will not succeed.

# 8 Migrate from Pro Glasses 2

## 8.1 Projects/participants/calibrations

Tobii Pro Glasses 3 does not have the concept of projects or participants, but if your client has these concepts the information can be stored as custom metadata in the recordings using the metadata interfaces in the recorder object. There is no support in Tobii Pro Glasses 3 to store participant/project data on the SD card except as a part of a recording.

## 8.2 Live view

Pro Glasses 2 supported two different protocols for live viewing. The first protocol that was supported was MPEG-TS over UDP with gaze data streaming over a persistent HTTP stream. In more recent firmware releases support was added for RTSP with gaze transmitted as a data channel in the RTSP stream.

Tobii Pro Glasses 3 supports RTSP with gaze transmitted in data channels, but the structure of the gaze packages has changed to align with the rest of the Tobii Pro Glasses 3 API. In addition to RTSP, Tobii Pro Glasses 3 also supports streaming video and data over WebRTC.

## 8.3 Replay

Pro Glasses 2 provided access to the recording files using a file share with guest access (no password). In Tobii Pro Glasses 3, all files belonging to a recording are provided over HTTP.

## 8.4 Discovery

Tobii Pro Glasses 2 initially came with a custom discovery mechanism implemented by broadcasting custom UDP packages in a dedicated port. Later versions of the firmware also added support for discovery via Zeroconf. Tobii Pro Glasses 3 only supports discovery via Zeroconf. It uses a separate service type from Tobii Pro Glasses 2 to be able to easily identify if a device is Tobii Pro Glasses 2 or Tobii Pro Glasses 3.

## 8.5 REST API

The REST API in Tobii Pro Glasses 3 is syntactically very different from Pro Glasses 2 even if they are semantically similar. We have tried to keep the number of concepts down to reduce the complexity in the Tobii Pro Glasses 3 API and stick to standards as much as possible. There is no easy way to convert existing code to from Tobii Pro Glasses 2 to Tobii Pro Glasses 3.

## 9 Firmware upgrade

The recommended way to upgrade firmware for Tobii Pro Glasses 3 is to use Tobii Pro Glasses 3 controller application. This will always give you the latest available firmware. The controller application will download and cache the latest firmware whenever it has access to internet and allow firmware upgrade also in offline mode. If your workflow requires that you update the firmware with a fully automated process, or to a version other than the latest available, it is possible to do that via the upgrade API. Please contact Tobii if you need access to firmware upgrade files. Tobii reserves the right to retract previously released versions of the Tobii Pro Glasses 3 firmware without prior notice.

The following should be ensured before performing a firmware upgrade:

- The head unit should be connected to the recording unit
- Sufficient battery or external power connected

Manually, the upgrade can be performed using the following unlisted web page from the Example web client: `http://<g3-address>/upgrade.html`

This page also acts as a reference implementation for how to use the RestAPI to send a firmware upgrade file (\*.tg3) and monitor the progress. To initiate a firmware upgrade, send the following request

- POST `http://<g3-address>/upgradeFile`
- Body: the contents of the tg3-file

During the firmware upgrade, the `upgrade:progress` signal will be sent continuously during the process. The signal contains two numeric values in the range 0..1 that indicates upload and upgrade progress respectively. Upload progress will be sent frequently. Upgrade progress will be sent between each individual upgrade package that is processed on the device.

After a firmware upgrade, the `upgrade:completed` original will indicate if the upgrade was successful or not and after that the firmware will be restarted including the wireless access point.



Note that some firmware upgrades include an updated firmware for the head unit. These will take significantly longer to complete and there will be long periods of time without any new data pushed on the `upgrade:progress` signal. Flashing the head unit FPGA firmware typically takes several minutes.

If a head unit with a mismatching FPGA firmware is connected to a recording unit, the firmware in the head unit will be updated immediately and the system will be unresponsive for several minutes.

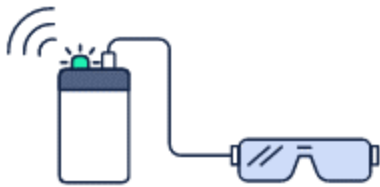
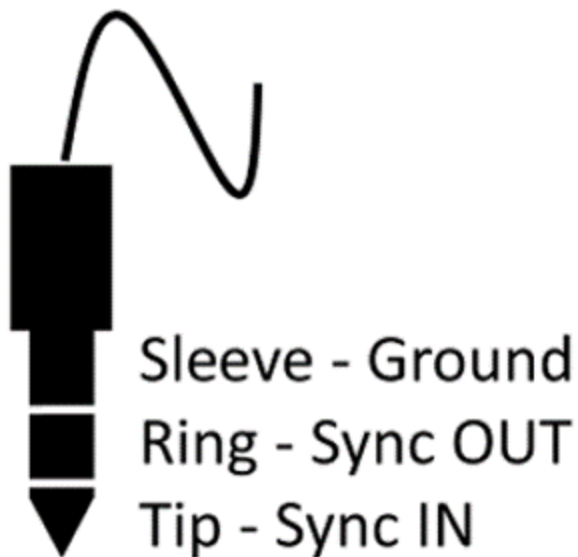
The firmware upgrade process does not support downgrading the firmware to an earlier release. If this is needed, please contact Tobii Support for assistance.



Please be aware that it is important that the firmware upgrade process is not interrupted. Make sure you have sufficient battery (and possibly an external power source) before initiating a firmware upgrade. If the firmware upgrade process is interrupted, it can leave the device in a bricked state that will require you to send it to Tobii for service.



## 10 TTL and sync port



### Setup description

The Pro Glasses recording unit connector must be 3-pin 3.5 mm jack with the following connections:

- Tip: Sync in - Receive events
- Ring: Sync out - Send out events
- Sleeve: Ground

As an example, if we want to receive TTL events in the glasses: the sleeve of the glasses port must be connected to the ground pin of the external device port and the tip must be connected to the pin of the external device port used to send the TTL events.

### Sync out - Send events from Tobii Pro Glasses 3

Tobii Pro Glasses 3 always send periodic 1-bit LVTTTL signals when a recording starts.

Events: The low state (0) corresponds to 0V and the high state (1) to 3V. Any change from low to high and vice versa will be timestamped in the event data stream:

```
{
  "type": "syncport",
  "timestamp": 0.28622,
  "data": {"direction": "out", "value": 1}
}
{
  "type": "syncport",
  "timestamp": 0.78622,
  "data": {"direction": "out", "value": 0}
}
```

Sequence: When the recording starts, a sequence of 1/0 is sent three times. Each state lasts 500ms. Starting 10 seconds after the first signal, a 1-second pulse is sent every 10 seconds.



### Sync in - Receive events in Tobii Pro Glasses 3

Tobii Pro Glasses 3 will timestamp any 1-bit TTL event sent to the glasses sync-in port.

Events: Any change from low (0: 0V) to high (1: from 2.5V to 3.3V) and vice versa detected in the sync-in port will be timestamped in the event stream:

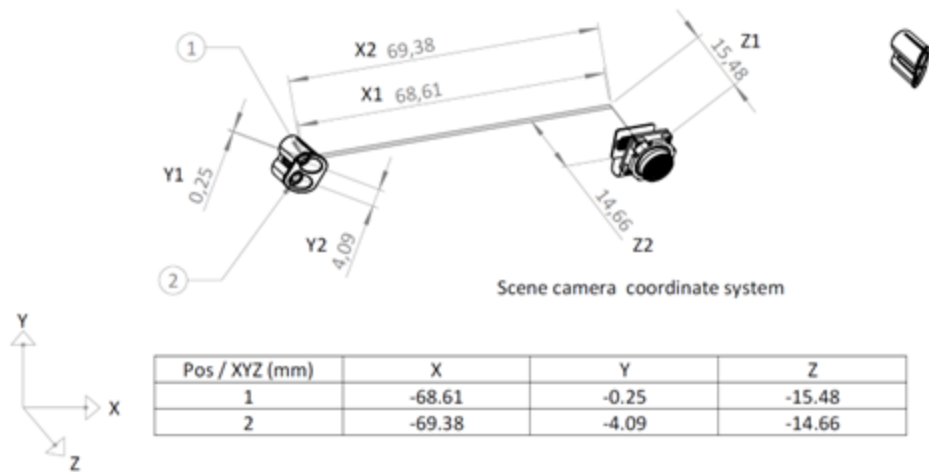
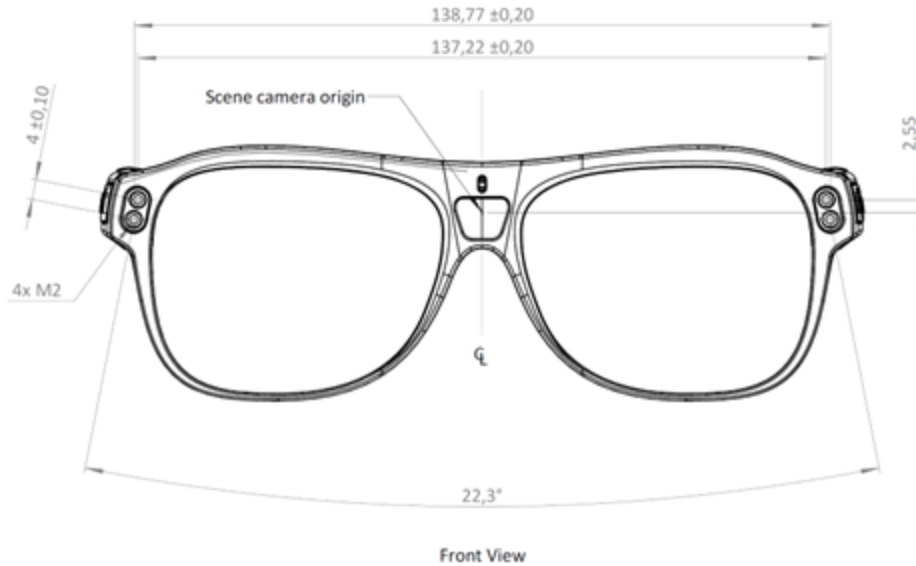
```
{
  "type": "syncport",
  "timestamp": 2.71828182,
  "data": {"direction": "in", "value": 1}
}
{
  "type": "syncport",
  "timestamp": 3.14159265,
  "data": {"direction": "in", "value": 0}
}
```



Note that the input voltage should not be higher than 3.3V.

# 11 Accessory mounts

Tobii Pro Glasses 3 has two mounting points for accessories on the left and right side of the front of the head unit.



These mounting points can be used for example, to attach IR markers for motion capture, fix the head unit to a helmet, or attach an external camera to the head unit.

# 12 IMU data

Tobii Pro Glasses 3 has an IMU sensor that can measure acceleration, rotation, and magnetic field. All three sensors are located close to the head unit scene camera. Each of the sensors must be calibrated in the environment that the data will be recorded since they are affected by parameters such as temperature.

## 12.1 Accelerometer

The accelerometer measures the acceleration of the head unit. Acceleration is measured along three axes in the head unit coordinate system. When at rest, the IMU sensor in the head unit will report an acceleration of  $\sim 9.8\text{m/s}^2$  upward (positive value in the y-component). This acceleration of 1G represents the normal gravity. Since gravity always points downwards, this can be used to measure tilt (roll and pitch) of a head unit in rest, but it can't be used to determine in which direction the glasses are facing. The accelerometer is sampled at  $\sim 100$  Hz.

## 12.2 Gyroscope

The gyroscope measures rotation around three axes.

- A yaw-movement (shaking of the head sideways) will give a rotation around the Y-axis.
- A pitch-movement (nodding the head up/down) will give a rotation around the X-axis.
- A roll-movement (tilting the head to the side) will give a rotation around the Z-axis

Rotation is measured around each axis in degrees per second and is sampled at  $\sim 100$  Hz.

## 12.3 Magnetometer

The magnetometer measures the strength of the surrounding magnetic field along three axes. This can be used to estimate the orientation of the glasses head unit. Keep in mind that the magnetometer will be affected by electromagnetic emissions from electronic equipment that is held near the head unit.

Magnetometer data is measured in microtesla ( $\mu\text{T}$ ) and is sampled at  $\sim 10$  Hz.

In the IMU signal, the samples can contain one or more readings from the three sensors depending on if they have been sampled at the same timestamp or not. For samples of the IMU data packages, read appendix section [IMU Data](#).

## 12.4 Coordinate system for IMU

Starting from firmware version 1.29, the IMU data is rotated and aligned with the Head Unit coordinate system, so that the data is expressed in the same coordinate system as the gaze data. Prior to this release, a user who wants to combine the data had to adjust the data manually.

The following matrices can be used to transform IMU data recorded prior to FW 1.29 to the Head Unit coordinate system:

Accelerometer and gyro	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -\cos(12^\circ) & \sin(12^\circ) \\ 0 & \sin(12^\circ) & \cos(12^\circ) \end{bmatrix}$
Magnetometer	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -\cos(12^\circ) & \sin(12^\circ) \\ 0 & \sin(12^\circ) & \cos(12^\circ) \end{bmatrix}$

These matrices handle both the fact that the IMU sensor is mounted with a 12 degree angle compared to the scene camera, and that some of the axis in the IMU sensor are flipped.

The exact location of the IMU sensor in the glasses 3 head unit is [0,0.13,-4.04]. This means that the IMU is 4.04 mm behind and 0.13mm above the center of the scene camera.

# Appendix A File structure on the SD card

Every recording is stored in a separate folder in the root of the SD card. The folder name defaults to the UTC time of the start recording operation but can be changed while the recording is in progress.

Rearranging or modifying the files in the recording folder could lead to problems replaying the recording using Tobii Pro Glasses 3. It could also lead to problems when importing the recording into Tobii Pro Lab for later analysis.

## A1 Example recording.g3 file

```
{
  "uuid":"03e9dfbf-83b8-45dd-8ee7-6fa28b0ba760",
  "name":"My First Recording",
  "version":1,
  "duration":12.767,
  "created":"2020-03-17T11:41:56.852577Z",
  "timezone":"Europe/Stockholm",
  "meta-folder":"meta",
  "scenecamera":{
    "file":"scenevideo.mp4",
    "camera-calibration": {
      "position":[0.0, 0.0, 0.0],
      "rotation":[
        [-0.999,0.014,-0.0207],
        [-0.0139,-0.999,-0.0131],
        [-0.0209,-0.0128,0.999]
      ],
      "focal-length":[917.21,916.74],
      "skew":0.0,
      "principal-point":[961.65,514.30],
      "radial-distortion":[-0.0511,0.0758,-0.0480],
      "tangential-distortion":[-0.000331,0.00000810]
    },
    "snapshots":[
      {
        "file":"snap0.jpg",
        "time":0.970396335
      }
    ]
  },
  "eyecameras":{
    "file":"eyevideo.mp4",
  },
  "gaze":{
```

```

    "file": "gazedata.gz",
    "samples": 641,
    "valid-samples": 623
  },
  "events": {
    "file": "eventdata.gz"
  }
}

```

## A2 Recordings folder

In each recording folder there is one main "recording.g3" information file which describes the recording and its current state.

The file uses utf-8 encoding of non ascii characters. The file has no byte order mark (BOM).

It consists of a JSON object with the following structure:

Name	Type	Description
uuid	string	A unique UUID representing the recording
name	string	The descriptive name of the recording (can be changed through the API using the "visible-name" property)
version	integer	The version of the recording.g3 file. Version will be increased when breaking changes occur (element name or element type changes). New elements will not cause version update
duration	double	The length of the recording in seconds
created	string	UTC date and time of recording start in ISO8601 format
timezone	string	The configured time zone at recording start
scenecamera	JSON object	Scene Camera Object (see below)
eyecameras	JSON object	Eye Camera Object (see below)
gaze	JSON object	Gaze Object (see below)
imu	JSON object	IMU Object (see below)
events	JSON object	Events Object (see below)
meta-folder	string	The name of the metadata folder

### A2.1 Scene Camera Object

Name	Type	Description
file	string	The name of the scene camera video file or <b>null</b> if no video was available
camera-calibration	JSON object	A Camera Calibration Object for the scene camera (see below)
snapshots	array	Array of Snapshot Objects (see below)

The scene camera file is an MP4 file with the following properties:

Stream 0 - video	
Codec	h.264 - MPEG-4 AVC

Stream 0 - video	
Video resolution	1920x1080
Frame rate	25 fps
Stream 1 - audio	
Codec	MPEG Audio layer 1 (mp3)
Channels	Mono
Sample rate	24000 Hz
Bits per sample	32
Bitrate	96 kb/s

## A2.2 Eye Camera Object

Name	Type	Description
file	string	The name of the eye camera video file or null if no video was available

## A2.3 Gaze Object

Name	Type	Description
file	string	The file name of the gaze data file
samples	integer	Total number of gaze samples in the gaze data file
valid-samples	integer	Total number of valid gaze samples in the gaze data file. A gaze sample is considered valid if at least one eye was successfully tracked

## A2.4 Camera Calibration Object

Name	Type	Description
position	double array	3D Position of the camera
rotation	double matrix	The 3x3 rotation matrix
focal-length	double array	2D Focal Length
skew	double	Camera skew
principal-point	double array	2D Principal Point
radial-distortion	double array	3D Radial Distortion
tangential-disortion	double array	2D Tangential Distortion
resolution	int array	Pixel resolution

## A2.5 Snapshot Object

Name	Type	Description
file	string	The file name of the snapshot
timestamp	double	Seconds in the recording the snapshot was taken



## A2.6 Events Object

Name	Type	Description
file	string	The file name of the events data file

## A2.7 IMU Object

Name	Type	Description
file	string	The file name of the IMU data file

## A2.8 Meta folder

The folder "meta" contains additional metadata that can be added through the recorder-object in the API.

The following files will always be stored in the metadata folder when the recording is started:

Name	Content	Description
RuVersion	string	The Tobii Pro Glasses 3 firmware version that was used when the recording was created
RuSerial	string	The Tobii Pro Glasses 3 serial number of the unit that created the recording
HuSerial	string	The serial number of the head unit that was used when the recording was created

Additional metadata stored by the client during the recording will be stored as files with the specified meta data key as the file name.

## A3 On disk data format

Data files in the recording folder (gaze, events, IMU) all share the same basic structure.

- Text files with LF as the line separator
- One JSON object per line
- Compressed with gzip
- File name ends with .gz

The JSON objects in each file also shares the following structure:

File name	Type	Description
type	string	The type of the JSON object
timestamp	double	Number of seconds into the recording that the data was sampled
data	JSON object	Specific content for each type of sample

### A3.1 Gaze data

If an eye is invalid, the eye object is empty. If both eyes are invalid, the data object is empty.

Field name	Type	Description
gaze2d	double array	Horizontal (x) and vertical (y) position of the gaze sample in normalized video coordinates (0-1). 0,0 corresponds to top left corner. 1,1, corresponds to bottom right corner.
gaze3d	double array	Position of the vergence point of the left and right gaze vector relative to the scene camera. Measured in millimeters.
eyeleft.gazeorigin eye-right.gazeorigin	double array	Position of the eye relative to the scene camera. IPD (inter pupil distance) can be calculated as the distance between the left and right gaze origin
eyeleft.gazedirection eye-right.gazedirection	double array	The estimated gaze direction in 3D. The origin of this vector is the gazeorigin for the respective eye.
eyeleft.pupildiameter eye-right.pupildiameter	double	Diameter in millimeter of the pupil.

Example:

Gaze sample with valid gaze for both eyes:	<pre>{   "type": "gaze",   "timestamp": 10.21,   "data": {     "gaze2d": [0.468, 0.483],     "gaze3d": [37.543, -18.034, 821.265],     "eyeleft": {       "gazeorigin": [28.799, -7.165, -23.945],       "gazedirection": [0.0463, -0.0337, 0.998],       "pupildiameter": 2.633,     },     "eyeright": {       "gazeorigin": [-28.367, -5.353, -21.426],       "gazedirection": [0.0436, 0.00611, 0.999],       "pupildiameter": 2.782,     }   } }</pre>
--	---

Gaze sample with invalid gaze for left eye:	<pre> ...   "data": {     "gaze2d": [0.468, 0.483],     "gaze3d": [37.543, -18.034, 821.265],     "eyeleft": {},     "eyeright": {       "gazeorigin": [-28.367, -5.353, -21.426],       "gazedirection": [0.0436, 0.00611, 0.999],       "pupildiameter": 2.782     }   } ... </pre>
Gaze sample with invalid gaze for right eye:	<pre> ...   "data": {     "gaze2d": [0.468, 0.483],     "gaze3d": [37.543, -18.034, 821.265],     "eyeleft": {       "gazeorigin": [28.799, -7.165, -23.945],       "gazedirection": [0.0463, -0.0337, 0.998],       "pupildiameter": 2.633     },     "eyeright": {}   } ... </pre>
Gaze sample with invalid gaze for both eyes:	<pre> ...   "data": {} ... </pre>

### A3.2 Sync Port Event Data

File name	Type	Description
direction	string	If change was on "in" or "out" port
value	int	The new value, 1 or 0.

Example:

Sync port sample when sync in changed to 1/high	<pre> {   "type": "syncport",   "timestamp": 1.13,   "data": {     "direction": "in",     "value": 1   } } </pre>
---	---

Sync port sample when sync out changed to 0/low	<pre>{   "type": "syncport",   "timestamp": 1.13,   "data": {     "direction": "out",     "value": 0   } }</pre>
---	--

### A3.3 Event Data

Field name	Type	Description
tag	string	A client specific tag name for the data structure format
object	JSON object	Client controlled data

Event example from a hypothetical simulator	<pre>{   "type": "event",   "timestamp": 4.13,   "data": {     "tag": "simulator telemetry",     "object": {       "joystick direction": "left",       "speed": 43.2,       "level": 3     }   } }</pre>
---	--

### A3.4 IMU Data

IMU data is taken directly from one or more IMU sensors. These sensors may work in different frequencies so each IMU sample may contain values from one or more individual sensors.

Field name	Type	Description
accelerometer	double array	Acceleration in three dimensions in the coordinate system of the head unit. The values are in m/s <sup>2</sup> . When the head unit is still, the accelerometer should indicate ~9.8 downward along the z-axis (obey gravity, it's the law)
magnetometer	double array	Measurement of the magnetic field near the head unit. This can be used to determine the orientation of the head unit. The values are in micro tesla.
gyroscope	double array	The gyroscope measures the rotation of the head unit in three dimensions. The values are in degrees/second.

Example:

IMU sample with only data from only one sensor	<pre>{   "type": "imu",   "timestamp": 1.23,   "data": {     "magnetometer": [-0.0418, 0.229, -0.196]   } }</pre>
Sync port sample	<pre>{   "type": "imu",   "timestamp": 1.33,   "data": {     "accelerometer": [-0.0427, -0.920, 0.472],     "gyroscope": [2.601, 0.0822, -0.179]   } }</pre>



---

Copyright ©Tobii AB (publ). Not all products and services offered in each local market. Specifications are subject to change without prior notice. All trademarks are the property of their respective owners.

## Support for your Tobii product

### Get help

Visit [Tobii Connect](#) for help with your Tobii device. It contains the latest information about contacting Customer Care, documentation, courses, articles, and more.

### Contact your solution consultant or reseller

For questions or issues with your product, contact your Tobii sales representative or authorized reseller for assistance. They are most familiar with your personal setup and can help you with tips and product training.